Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 87–105

www.elsevier.com/locate/entcs

Higher-Order Nets for Mobile Policies

Kathrin Hoffmann[1,5]

ISTI, Technical University Berlin, Germany

Till Mossakowski [2,5]

BISS, University of Bremen, Germany

Francesco Parisi-Presicce [3,4,5]

Dip. di Informatica, Università di Roma "La Sapienza", Italy
ISE Department, George Mason University, USA

Abstract

Since the early 80's the combination of Petri nets and rule-based transformations has been extensively researched to obtain new concepts and results. In this paper we consider rules as tokens leading to the concept of higher-order nets for mobile policies. The rules are used on the one hand for the specification of policy rules and on the other hand for the modification of policy rules, i.e. for the definition of new rules by reusing existing rules. So the higher-order net models distribution and modification of policy rules in a systematic and structured way. We give HasCasl-specifications of rules and (local) transformations in the sense of the double-pushout approach and illustrate our concept by a small system inspired by the case study of a tax refund process [1].

Keywords: algebraic higher-order nets, mobile policies, rule-based transformations, algebraic specifications, case study: tax refund process.

[1] Email: hoffmann@cs.tu-berlin.de
[2] Email: till@tzi.de
[3] Email: parisi@di.uniroma1.it
[4] Email: fparisi@ise.gmu.edu
[5] Partially supported by the European Union under Research and Training Network SegraVis

doi:10.1016/j.entcs.2005.02.007

1 Introduction

A policy is a set of rules which controls the behavior of complex systems. In [10] a policy framework based on graph transformation is presented, which provides an intuitive visual formalism for the manipulation of policy-based systems. The policy framework is defined by a type graph and sets of policy rules, positive and negative constraints.

Mobile policies [2,5] are policies that can move along with the application or data that they refer to. Mobile policies can either supplement or override local policies, and can be used either to regulate access to the local resources (to protect the host) or to constraint use or access to the mobile code (to protect the guest). In a distributed environment, applications migrate from site to site and the relative policy can migrate with the application it refers to, thus allowing each site to avoid locally storing policies for all possible applications. A framework in which mobile policies are attached to the relative application also facilitates the development of new applications and places the responsibility for the application-specific policy on the application designer.

Mobile policies may also need to be modified in prearranged ways to adapt to external requirements of specific domains. For example, certain local laws may require or forbid certain behavior of all applications executing locally (restrictions on the use of encryption or on the length of the key in a cryptosystems, additional requirements to protect privacy) and therefore the constraints imposed by the policy may need to be adapted in moving from site to site.

We investigate how the distribution, the migration, and the modification of mobile policies can be modeled by using Algebraic Higher-Order (AHO) nets [8], a high-level net class integrating Petri nets and the higher-order specification language HasCasl [16]. Due to the higher-order features, graphs and rules are allowed to be dynamic objects in AHO-nets and the behavior of an AHO-net simulates the modification needed to achieve the flexibility of adapting objects.

For our purpose, the AHO-net gives an overview of the different locations where the mobile policies could reside. Furthermore, the coupling of a set of rules that are used to modify policy rules with certain locations that have the authority to modify the policy rules is given by the net topology. The behavior of an AHO-net simulates the application of a rule to a policy rule and describes the modification of the policy rule in order to achieve a more appropriate one. In this paper policy rules are used for the specification of access control [15]. Apart from this, the concept has interesting applications in all areas where individual rules are modified while the system is running.

Rules and transformations are formalized on a rigorous mathematical foundation in the context of high-level replacement systems [6], a categorical generalization of the concept of graph transformation systems to other kinds of structures based on the double-pushout approach. A high-level replacement system is defined by an arbitrary category and a distinguished class of morphisms used to form rules, i.e. rules in the double-pushout approach are given as a span of two morphisms, and its application is achieved by two pushouts. Think of these rules as replacement systems, where the left-hand side of the rule is replaced by the right-hand side. Moreover, we reuse policy rules, i.e. we modify policy rules in the sense of inheritance [14].

The strong relationship between the area of Petri nets and graph transformation systems has been researched in a series of papers. On the one hand, looking at Petri nets from the perspective of graph grammars, it is quite natural to regard them as grammars acting on discrete graphs. In this way transitions can be represented by graph rules and the application of such a rule simulates the token game (see e.g. [3,4,11]). On the other hand the concept of high-level replacement systems [6] was the starting point to obtain new results for the area of Petri nets. The instantiation of high-level replacement systems to Petri nets leads to the concept of net transformation systems [13,17]. The basic idea behind net transformation systems is the stepwise development of systems in the framework of Petri nets. In this paper the concept of AHO-nets integrates rules and transformations into the data type part. As a consequence the behavior of a system describes the transformation of objects. The paradigm "nets as tokens" has been introduced by Valk in order to allow for nets as tokens within a net (see [18,19]). There are interesting applications in the area of workflow, agent-oriented approaches, or open system networks. We propose the new paradigm "rules as tokens", where in contrast rules as tokens are considered.

Technically, we extend the HasCasl-specification of rule-based modifications in [9] by specific operations for the modification of mobile policies. Here, we use the approach given in [14] to obtain a suitable specification of the modification of rules. We use this specification as the data type part of AHO-nets to denote the application of rules in the net inscriptions. Then the behaviour of the AHO-net simulates the modification of rules to obtain a new and more appropriate policy rule.

The advantage of our approach is twofold. On the one hand the AHO-net manages the distribution of rules in a systematic and structured way. Large sets of rules are divided into smaller ones, which are locally bound to some transitions. On the other hand AHO-nets are flexible in respect of the replacement of rules. Formally, we exchange rules by exchanging the corresponding

tokens to realize other kinds of transformations, while the system net is fixed.

To demonstrate our approach in more detail we give an example of a tax refund process in Section 2. Afterwards, we present the technical background, i.e. the HasCasl-specification of rules and (local) transformations (Section 3) and a light-weight introduction into AHO-nets (Section 4). The last section summarizes the paper and discusses some future work. The HasCasl specifications involved have been checked with the Heterogeneous Tool Set [12].

2 Example: Tax Refund Process

In order to illustrate the concepts described in Section 3 and Section 4 we present a small system inspired by the case study of a tax refund process given in [1]. The main idea of our example is to model mobile policies which move around between different companies. The policy rules are not fixed once and for all, because each company expects specific policy rules. Our example is restricted in the sense that we do not take into account all aspects of the policy framework presented in [10]. In this paper we specify policy rules that build the accepted system states, and assume that the policy rules are built over a given type graph. Furthermore, we do not focus on the application of policy rules to the actual state of an object. In [7] we have studied high-level object systems where the application of rules to objects is modeled by corresponding operations. Thus, the system presented in this paper can be extended by these concepts, i.e. the set of rules can be extended to a graph grammar.

The example deals with a tax refund process which is a simplified version of the workflow introduced in [1]. The workflow representing the tax refund process in company $C1$ consists of four tasks to be executed sequentially:

- Task $T1$: A clerk prepares a check for a tax refund.
- Task $T2$: A manager can approve or disapprove the check. This task must be performed by two managers.
- Task $T3$: The decisions of the managers are collected and the final decision is made by a manager. Her/his decision is a consequence of the outcome of task $T2$, i.e. (s)he does not decide about the tax refund.
- Task $T4$: A clerk issues if both managers approved or voids if one manager disapproved the check on the result of task $T3$.

By contrast, the tax refund process in the company $C2$ is altered in task $T2$ and task $T4$, while Task $T1$ and Task $T3$ are left unchanged:

- Task $T2$: A manager can approve or disapprove the check. This task must

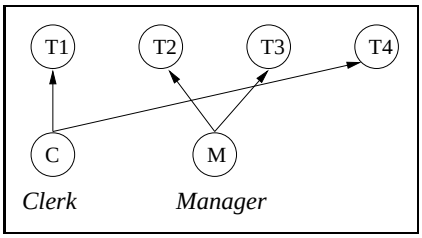


Fig. 1. Role authorization

be performed by one manager.

- Task $T4$: A clerk issues if the manager approved or voids if the manager disapproved the check on the result of task $T3$.

In Fig. 1 each task is related to a role which can execute the task, e.g. the role *Clerk* can execute task $T1$ and task $T4$.

Our first goal is a representation of the system level as an AHO-net (see Section 4), so that the system shows on the one hand the distribution of policy rules, and on the other hand the coupling of rules for the modification of policy rules to certain transitions. Thus, the firing behavior of the AHO-net describes the migration and local transformation of the policy rules. In Fig. 2 we sketch a solution for the example of the tax refund process. The initial marking and the net inscriptions of the AHO-net in Fig. 2 are built over the HasCasl-specification `LocalTransformation[LGraphCategory]` and the corresponding algebra $\mathcal{A}$ which will be explained in Section 3. There are four different locations where the policy rules can stay: the company $C1$, the company $C2$, and during the migration processes, between $C1$ and $C2$, or $C2$ and $C1$. Each location becomes represented by its own place in the AHO-net in Fig. 2. The initial marking consists of the policy rules $PolRules_{C1}$ of company $C1$ and specific rules for the modification of policy rules.

Policy rules may move around, which means they might leave and enter the company $C1$ and they might leave and enter the company $C2$. The mobility aspect of the policy rules is modeled by transitions termed in an obvious way in our system net in Fig. 2. While the policy rules are moving around they have to be changed in a certain way using the concept of inheritance (see Section 3). For this reason there are other kinds of rules, $p_{13} - p_{15}$ and $p_{13}^{-1} - p_{15}^{-1}$, to guarantee the modification of policy rules. Here these rules are used as resources, so they are bound to corresponding transitions.

Thus, the object level consists of two different kinds of objects: policy rules and rules for the modification of policy rules. In the following we will explain the policy rules $PolRules_{C1}$ of company $C1$ in more detail. Although they are based on the rules given in [10], in this paper we use the double-pushout

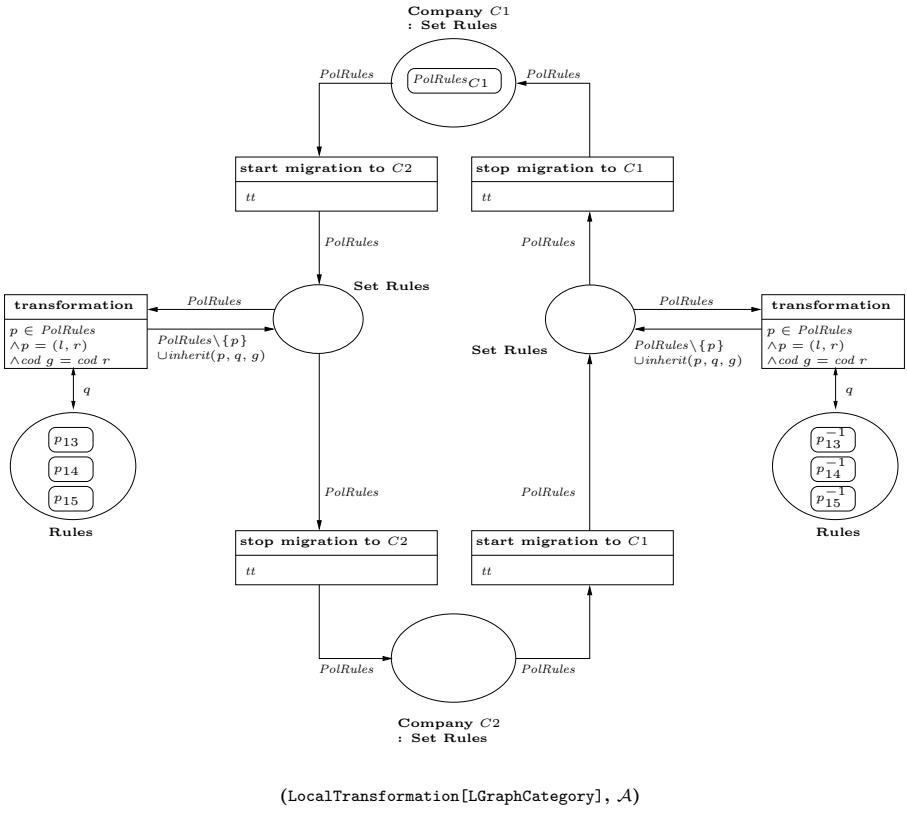


Fig. 2. System net of the tax refund process

approach instead of the single-pushout approach and we have to take care of the gluing condition. For simplicity reasons we avoid negative application conditions, i.e. we cannot distinguish between the users which are involved in the tax refund process.

The set of policy rules of company $C1$ is given by $PolRules_{C1} = \{p_1, \ldots, p_9\}$. The rule p_1 (see Fig. 3) creates a new check by a user associated to the role *Clerk*. A user is represented by a node of type U. The two loops of the *check* node indicate that the recommendation for a tax refund has to be performed by two managers. The rules p_2 and p_3 realize the process of task $T2$ (see Fig. 4), i.e. a manager approves or disapproves the check. In detail, a loop from the *check* node is deleted and an edge between the *user* node and the *check* node labeled with the recommendation is created. Thus, these rules are only applicable if there is a loop attached to the *check* node. The rules p_4, p_5 and

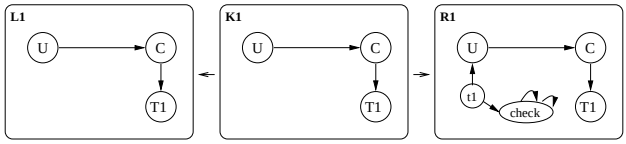


Fig. 3. Rule p_1 for task $T1$ in company $C1$

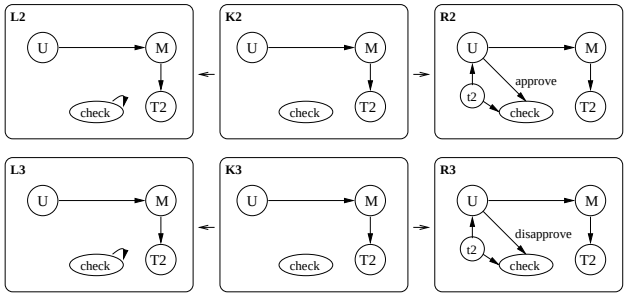


Fig. 4. Rules p_2 and p_3 for task $T2$ in company $C1$

p_6 realize the process of task $T3$ (see Fig. 5), i.e. the collection of the decisions. The two edges which model the recommendations of the two managers are deleted and a loop of the *check* node is created labeled by *issue* if both managers approve and by *void* if one of the managers disapproves. In all three cases the manager does not decide directly, because the decision is based on the previous recommendations. The rules p_7 and p_8 realize the process of task $T4$ (see Fig. 6), i.e. a clerk issues or voids the check. The end of the workflow for this check is indicated by changing the color of the *check* node and deleting the corresponding loop. Finally, the tax refund process is finished using rule p_9 (see Fig. 6) by deleting the *check* node and all connected nodes t1-t4 and adjacent edges.

The set of policy rules described above may move around between the two companies. Because each company expects specific policy rules, some rules have to be modified during the migration. In detail the following policy rules of company $C1$ have to be modified to respect the requirements of company $C2$:

- preparation of the check (see rule p_1 in Fig. 3)
- approval of the check (see rule p_2 in Fig. 4)
- disapproval of the check (see rule p_3 in Fig. 4)

To integrate the modification of policy rules into our model, we need an operation to achieve new rules by reusing existing ones. Here we use the approach of local transformation as presented in [14] to get a new rule which coincides with the left hand side of the "old" rule, but which has a different

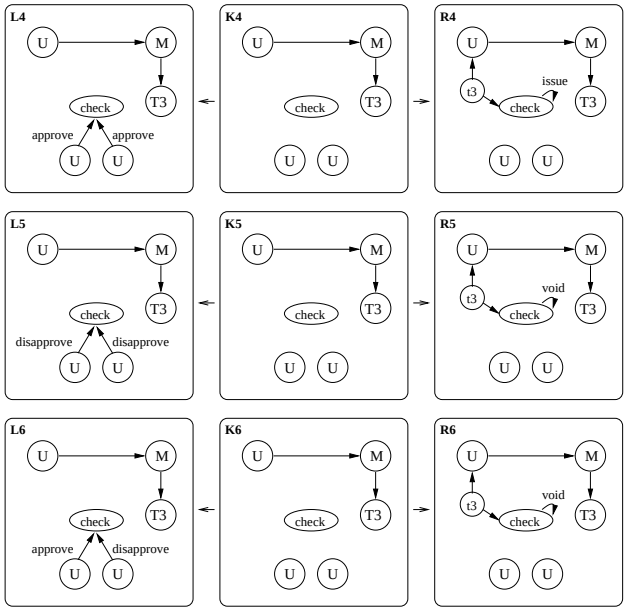


Fig. 5. Rules p_4, p_5 and p_6 for task $T3$ in company $C1$

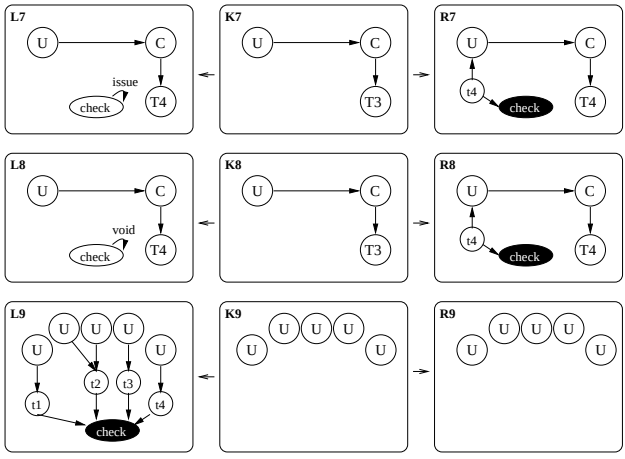


Fig. 6. Rules p_7, p_8 and p_9 for task $T4$ in company $C1$

right hand side and (in general) a different interface part. For a detailed explanation we refer to Section 3.

The modification of the rule p_1 (see Fig. 3) is attained via the rule p_{13} (see Fig. 7). Here we use the transition *transformation* of the AHO-net in Fig. 2. First, the net inscriptions in the environment of the transition are evaluated, i.e. the variable *PolRules* is assigned to the set of policy rules $PolRules_{C1}$, the

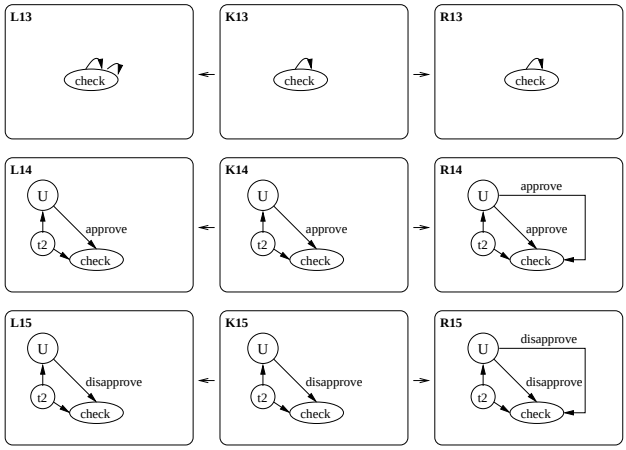


Fig. 7. Rules p_{13}, p_{14} and p_{15} for the modification of policy rules

variable p to the rule p_1, the variable q to the rule p_{13}, and the variable g to an occurrence morphism $g_1 : L \to G$. The firing condition $cod\ g = cod\ r$ requires $L = L_{13}$ and $G = R_1$. The transition *transformation* is enabled under this assignment, i.e. the evaluation of the net inscriptions is defined. Then the evaluation of the term $inherit(p, q, g)$ computes the modification of rule p_1 via the rule p_{13} using the concept of inheritance. We obtain the new policy rule p_{10} of company $C2$ depicted in Fig. 8. The rule p_{10} adds one loop to the *check* node because one manager has to approve or disapprove the check.

For the modification of the rule p_2 resulting the new rule p_{11} (see Fig. 9) we use a different variable assignment, i.e. the variable p is assigned to the rule p_2, the variable q to the rule p_{14}, and the variable g to an occurrence morphism $g_1 : L \to G$ so that $L = L_{14}$ and $G = R_2$. Then R_{11} is the object resulting from the direct transformation via p_{14} of R_2, K_{11} is the common part of C_1 and K_2, and L_{11} is just the unchanged left hand side of p_2 (see Fig. 9). The new rule $p_{11} = (L_{11} \leftarrow K_{11} \rightarrow R_{11})$ adds two edges with the same label *approve* to the *check* node, because the clerk issues if one manager has approved. Analogously, the modification of the rule p_3 (see Fig. 4) via the rule p_{15} (see Fig. 7) results in the policy rule p_{12} of company C_2 (see Fig. 10). Finally, the set of policy rules of company $C2$ consists of the rules p_4 - p_{12} (see Figs. 5, 6, 8, 9, and 10), where the three rules concerning the preparation of a check and the approval or disapproval of a check are modified.

3 Specification of Rule-Based Transformations

In this section we review the basic concepts of rules and (local) transformations in order to capture these concepts in HasCasl-specifications. The idea

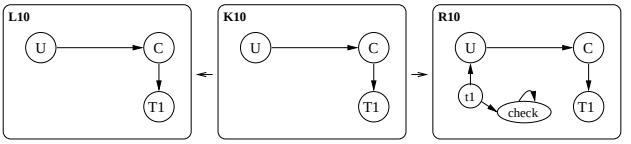


Fig. 8. Rule p_{10} for task $T1$ in company $C2$

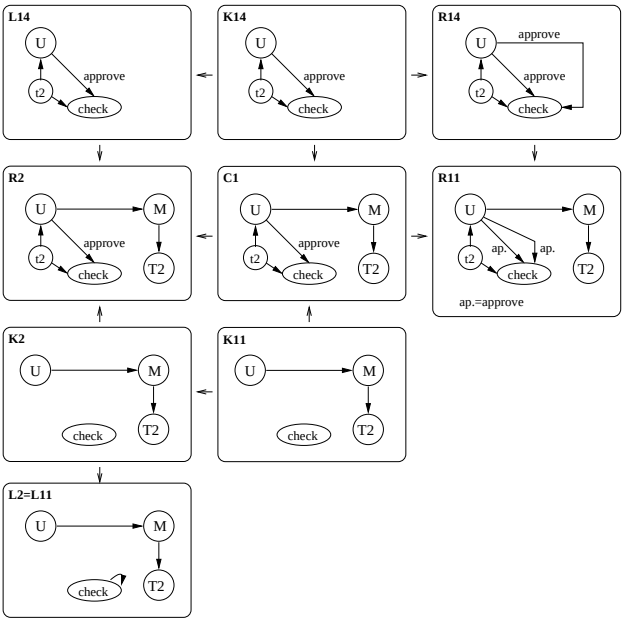


Fig. 9. Modification of p_2 via p_{14} to achieve p_{11}

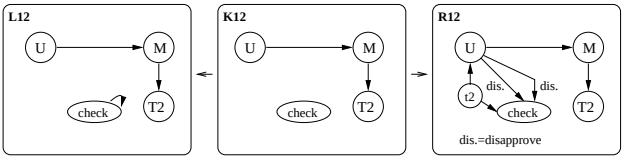
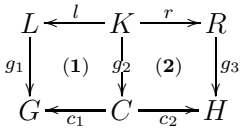


Fig. 10. Rule p_{12} for task $T2$ in company $C2$

of rules and transformations is to define any kind of system development and modification as an abstract rewriting. The application of a rule replaces the left-hand side of the rule by the right-hand side of the rule and is called transformation. In [6] the general description of rules and transformations has been introduced as a categorical generalization of graph transformations leading to the notion of high-level replacement systems. High-level replacement systems are formulated for an arbitrary category with a distinguished class of morphisms which are used in the description of rules.

Let $\mathbf{C}$ be a high-level replacement category, i.e. a category that comes with a distinguished class of morphisms $\mathcal{M}$, and selected pushouts along morphisms in $\mathcal{M}$. A rule $p = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ in $\mathbf{C}$ consists of the objects L, K and R, called left-hand side, interface (or gluing object) and right-hand side resp., and two morphisms $K \stackrel{l}{\rightarrow} L$ and $K \stackrel{r}{\rightarrow} R$ belonging to $\mathcal{M}$. Given a rule $p = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ and an object C, called context object, together with a morphism $K \stackrel{g_2}{\rightarrow} C$, a direct transformation $G \stackrel{p}{\Longrightarrow} H$ from an object G to an object H is given by two pushout diagrams **(1)** and **(2)** in the category $\mathbf{C}$. The morphisms $L \stackrel{g_1}{\rightarrow} G$ and $R \stackrel{g_3}{\rightarrow} H$ are called occurrences of L in G and R in H, respectively. Given a rule $p = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$, an object G and an occurrence $L \stackrel{g_1}{\rightarrow} G$ of L in G, the rule p is applicable to G via $L \stackrel{g_1}{\rightarrow} G$ if the following two conditions are satisfied:



(i) There are a unique object C and morphisms $K \stackrel{g_2}{\rightarrow} C$ and $C \stackrel{c_1}{\rightarrow} G$ such that the diagram **(1)** becomes a unique pushout. In this case C is called pushout complement of L w.r.t. G and K in **(1)**.

(ii) There is an object H which is the selected pushout of morphisms $K \stackrel{r}{\rightarrow} R$ and $K \stackrel{g_2}{\rightarrow} C$ in diagram **(2)** (pushout construction).

If both conditions are satisfied, a direct transformation $G \stackrel{p}{\Longrightarrow} H$ can be constructed and H is uniquely determined.

Example 3.1 The category of (labeled) directed graphs with the distinguished class of injective, colour preserving graph morphisms has been checked to be a high-level replacement category (see [6]) with the capacity to guarantee Church-Rosser and Parallelism Theorems for high-level replacement. An example of a direct transformation can be found in Fig. 9 where the rule p_{14} (see Fig. 7) is applied to the object R_2.

We use the higher-order specification language HasCasl [16] (an extension of higher-order logic with partial functions and subsorting) to formalize rules and transformations. In this way we can use rules as tokens in AHO-nets (see Section 4). Based on a specification of categories in HasCasl [6], we have specified transformations via the double-pushout approach with a HasCasl-specification Transformation[HLRCategory] (see Fig. 11). The notation $M < Mor$ introduces M as a subsort (roughly corresponding to a subset) of sort *Mor*. Note that *transform* is a partial function, and *def transform(p,g1)*

[6] We can only present part of the involved specifications here. All the specification can be found under http://www.cofi.info/Libraries/.

spec HLRCategory = Pushout[Category]
reveal ***sorts*** Ob, Mor, M, ***ops*** $id, dom, cod, \_o\_$
then
sort $M < Mor$

spec Transformation[HLRCategory] = Pushout[Category]
then
pred $POComplement : Mor \times Mor \times Ob$
forall $o : Ob;\ f, h : Mor$
$\bullet\ POComplement(f, h, o) \Leftrightarrow$
$\exists g, k : Mor \bullet (h, k) = f\ pushout\ g\ \wedge\ dom\ k = o$

type $Rules = \{(l, r) : M \times M \bullet dom\ l = dom\ r\}$

ops $transform : Rules \times Mor \rightarrow ?\ Rules;$

forall $p : Rules;\ g1 : Mor$
$\bullet\ let\ (l, r) = p\ \text{in}$
$def\ transform(p, g1) \Leftrightarrow$
$\exists g2, g3, c1, c2 : Mor \bullet POComplement(l, g1, dom\ c1)$
$\wedge\ (\forall o1, o2 : Ob \bullet POComplement(l, g1, o1)$
$\wedge\ POComplement(l, g1, o2) \Rightarrow o1 = o2)$
$\wedge\ (g1, c1) = l\ pushout\ g2$
$\wedge\ (g3, c2) = r\ pushout\ g2$

$\bullet\ let\ (l, r) = p\ \text{in}$
$def\ transform(p, g1) \Rightarrow$
$\exists g2, g3, c1, c2 : Mor \bullet (g1, c1) = l\ pushout\ g2$
$\wedge\ (g3, c2) = r\ pushout\ g2$
$\wedge\ transform(p, g1) = (c1, c2)$

Fig. 11. Specification of graph morphisms in HasCasl

states that p is applicable with occurrence morphism $g1$. The first axiom in Fig. 11 specifies the domain of definition for *transform*, while the second axiom specifies its effect when defined.

Example 3.2 In the appendix, we give an instantiation of the HasCasl-specification HLRCategory by the HasCasl-specification LGraphCategory i.e. labeled directed graphs. This specification relies on vocabularies for nodes, edges, and labels. These vocabularies are given by type variables in the first place, and later on by sorts (when forming the category), since a category needs to have definite sorts of objects and morphisms. Type constructors are not involved. Actually, the only freedom in the models is the interpretation

spec LocalTransformation[HLRCategory] =

Transformation[HLRCategory]

then

ops $inherit : Rules \times Rules \times Mor \to ? Rules;$

forall $p1, p : Rules;\ g1 : Mor$

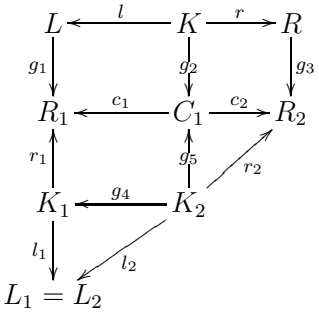
- $let\ (l1, r1) = p1\ \wedge\ (l, r) = p\ \text{ in}$
  $def\ inherit(p1, p, g1) \Leftrightarrow cod\ g1 = cod\ r1$
  $\wedge\ def\ transform(p, g1)$
  $\wedge\ def\ r1\ pullback\ c1$

- $let\ (l1, r1) = p1\ \wedge\ (l, r) = p\ \text{ in}$
  $def\ inherit(p1, p, g1) \Rightarrow$
  $\exists g4, g5, c1, c2 : Mor \bullet transform(p, g1) = (c1, c2)$
  $\wedge\ r1\ pullback\ c1 = (g4, g5)$
  $\wedge\ inherit(p1, p, g1) = (l1\ o\ g4, c2\ o\ g5)$

Fig. 12. Specification of inheritance in HasCasl

of these sorts. Typical choices will be the set of integers or the set of strings. Once this choice has been made, the remaining parts of the models are determined uniquely up to isomorphism, and hence a canonical model for the specification can be obtained.

To define new rules by reusing existing rules we use the approach given in [14], where different concepts are presented for the modification of one rule by another one. In this paper we describe the more general form of inheritance in more detail. Let a rule $p_1 = (L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1)$ be given. Then the modification of the right-hand side R_1 via a rule $q = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is illustrated in the upper part of the diagram, where R_2 is the object resulting from the direct transformation via q of R_1, K_2 is the common part of C_1 and K_1, and L_2 is just the unchanged left-hand side of p_1. Then the new rule p_2 is given by $p_2 = (L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2)$ with $l_2 = l_1 \circ g_4$ and $r_2 = c_2 \circ g_5$. Based on a specification of Transformation[HLRCategory] we have specified the concept of inheritance with a HasCasl-specification LocalTransformation[HLRCategory] (see Fig. 12) following the construction above.



Example 3.3 Analogously to the instantiation given in Example 3.2 we ob-

tain the specification LocalTransformation[LGraphCategory] by an instantiation of the HasCasl-specification HLRCategory by the HasCasl-specification LGraphCategory. An example of the modification of one rule by another rule using the concept of inheritance is depicted in Fig. 9. Thus, we have a specific operation $inheritance_{\mathcal{A}} : \mathcal{A}_{Rules} \times \mathcal{A}_{Rules} \times \mathcal{A}_{Mor} \to \mathcal{A}_{Rules}$, so that $inheritance_A(p_2, p_{14}, g_1) = p_{11}$, where p_2 is the policy rule of company $C1$ (see Fig. 4), p_{14} is a rule for the modification of p_2 (see Fig. 7), g_1 is a suitable occurrence morphism, and p_{11} is the policy rule of company $C2$ (see Fig. 9).

4 Higher-Order Nets for Local Transformations

In this section we present the integration of Petri nets and local transformations into AHO-nets, a high-level net class where the tokens are not only simple data values but the data values may be of arbitrarily complex type according to the higher-order specification language HasCasl. In contrast to the definition of AHO-nets in [8] we here use AHO-nets with a fixed data type part.

An AHO-net N for mobile policies is a tuple

$$(\text{LocalTransformation}[\text{LGraphCategory}], \mathcal{A}, P, T, pre, post, cond, type)$$

with

- the data type part given by the HasCasl-specification of local transformations and a corresponding model $\mathcal{A}$ (see Section 3);
- the net structure given by a set of (typed) places P and a set of transitions T; the pre- and post domain functions $pre, post : T \to (T_\Sigma(X) \otimes P)^\oplus$ assign net inscriptions to each transition (we denote by $T_\Sigma(X)$ the set of terms with variables and by $T_\Sigma(X) \otimes P$ the set of all type consistent net inscriptions, and finally by $(\_)^\oplus$ the free commutative monoid over this set); the firing condition function $cond : T \to T_{\Sigma,unit}(X)$ assigns one predicate to each transition, which is a constraint to be respected.

An example of an AHO-net for mobile policies can be found in Figure 2.

After defining the structure of AHO-nets we are now ready to look at their behavior. But first we define the set $Var(t)$ of variables of a transition $t \in T$ as the set of all variables occurring in pre- and post domain and in the firing condition.

The marking determines the distribution of tokens. Formally the marking $M \in (\mathcal{A} \otimes P)^\oplus$ of an AHO-net N with the set of places P consists of data values, which are elements from a given higher-order model $\mathcal{A}$. For each place

all tokens must belong to a specified type. So the marking is an element of the free commutative monoid over the set of all type consistent data values. We need monoids to allow one or more tokens to be at one place.

Data values can be modified during the firing of transitions. A data value can be moved along a transition, if the firing conditions are fulfilled. The follower marking is computed by the evaluation of net inscriptions. Given an assignment $v : Var(t) \to \mathcal{A}$, the transition t is enabled in a marking M, iff

(1) $\forall (term, p) \in pre(t) \oplus post(t) : term \in dom\ v^\sharp_s$ for $s = type(p)$

(2) $cond(t) \in dom\ v^\sharp_{unit}$ with $\mathcal{A}_{unit} = \{*\}$

(3) $\hat{v}(pre(t)) \leq M$

where $term \in T_\Sigma(X)_s$ is a term of type s and the value of $term$ in $\mathcal{A}$ under the assignment v is $v^\sharp(term) \in \mathcal{A}_s$. Furthermore, $\hat{v} : (T_\Sigma(X) \otimes P)^\oplus \to (\mathcal{A} \otimes P)^\oplus$ is the extension of the term evaluation $v^\sharp : T_\Sigma(X) \to \mathcal{A}$ to terms and places. Then the follower marking after the firing of t is defined by $M' = M \ominus \hat{v}(pre(t)) \oplus \hat{v}(post(t))$.

In contrast to the definition in [8] the definition of AHO-nets is more adapted to the design of HasCasl. On the one hand the firing condition function assigns one predicate to each transition instead of a finite set of equations, and on the other hand we claim that a transition is enabled, if and only if the (partial) evaluation of the net inscriptions in the environment of the transition is defined for a given assignment (see conditions (1) and (2) above).

In [9] we have achieved different kinds of objects, i.e. we have demonstrated the use of HasCasl-specifications on the one hand of graphs, Petri nets, and Petri systems and on the other hand of rules and transformations in the sense of the double-pushout approach as different data type parts of AHO-nets. The main result of this contribution is a suitable HasCasl-specification for mobile policies.

5 Conclusion and Future Work

Summarizing, we have presented a powerful technique to model mobile policies using AHO-nets in order to achieve highly expressive models. We have reviewed the concept of rules and transformations in the sense of the double-pushout approach and the concept of local transformations and have transferred these concepts into HasCasl-specifications. Afterwards we have explained the structure and behavior of AHO-nets. We have illustrated the use

of AHO-nets for mobile policies through the example of the tax refund process. Our system describes the migration of policy rules from one company $C1$ to another company $C2$. Moreover, policy rules become modified during the migration process by specific rules, so that the application of these rules results in new rules matching the requirements of the companies. Thus, local transformations become effectively included into the system enabling the system to transform rules in a formal way.

The main advantage of using AHO-nets is their flexibility in respect of introducing new rules to the system. While the system level is fixed, we can add further policy rules and rules for the modification of policy rules by adding further tokens of type *Rules* to our model. Note that the structure of these rules can be different from the structure of the rules presented in Section 2.

An interesting aspect of future work is to integrate not only the specification of policy rules but also the other aspects of the policy framework presented in [10] into our system, i.e. the type graph, the set of positive and negative graphical constraints and the application of policy rules to build the actual system state.

In this paper we have used the concept of inheritance to modify policy rules. But there are other concepts, e.g. the concept of specialization where properties are added to policy rules or the concept of analogy where a policy rule becomes reused in a different context. Here the approach given in [14] is a good starting point to obtain a suitable specification.

References

[1] E. Bertino, E. Ferrari, E., and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Transactions on Information and System Security*, 2(1):65-104, 1999.

[2] S. Chapin, D. Faatz, S. Jajodia, and A. Fayad. Consistent Policy enforcement in distributed systems using mobile policies. *Data & Knowledge Engineering*, 43:261-280, 2002.

[3] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract Graph Derivations in the Double Pushout Approach. In *Proc. Dagstuhl Seminar on Graph Transformations in Computer Science*, LNCS 776, pages 86–103. Springer, 1994.

[4] A. Corradini and U. Montanari. Specification of Concurrent Systems: From Petri Nets to Graph Grammars. In G. Hommel, editor, *Quality of Communication-Based Systems*, pages 35-52. Kluwer, 1995.

[5] V. Doshi, A. Fayad, S. Jajodia, and R. MacLean. Using attribute certificates with mobile policies in electronic-commerce applications. In *Proc. 16th Annual Computer Security Applications Conference*, pages 298-307. IEEE Computer Society, 2000.

[6] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comp. Science*, 1:361–404, 1991.

[7] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In *Proc. 26th International Conference On Application and Theory of Petri Nets and Other Models of Concurrency*. submitted.

[8] K. Hoffmann and T. Mossakowski. Algebraic Higher-Order Nets: Graphs and Petri Nets as Tokens. In M. Wirsing, D. Pattinson, and R. Henicker, editors, *Proc. 16th International Workshop of Algebraic Development Techniques*, LNCS 2755, pages 253–267. Springer, 2003.

[9] K. Hoffmann and T. Mossakowski. Integration of Petri Nets and Rule-Based Transformations. Technical Report, Technical University Berlin, 2004.

[10] M. Koch, and F. Parisi-Presicce. Describing Policies with Graph Constraints and Rules. In H. Ehrig, H.-J. Kreowski,G. Rozenberg, editors, *Proc. 1th International Conference of Graph Transformation*, LNCS 2505, pages 223–238. Springer, 2002.

[11] M. Korff and L. Ribeiro. An attributed graph transformation approach to the behaviour of algebraic high-level nets. Extended abstract for the international Workshop on Graph Grammars, 1994.

[12] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen. 2005.

[13] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic high-level net transformation systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[14] F. Parisi-Presicce. On modifying high level replacement systems. In H. Ehrig, C. Ermel, and J. Padberg, editors, *Proc.Uniform Approaches to Graphical Process Specification Techniques*, ENTCS 44(4), pages 1-12. Elsevier, 2001.

[15] R.S. Sandhu. Role-Bases Access Control. *Advances in Computers*, volume 46. Academic Press, 1998.

[16] L. Schröder, T. Mossakowski, and C. Maeder. HasCasl – Integrated functional specification and programming. Language summary. 2004. Available at www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HasCASL.

[17] M. Urbášek. *Categorical Net Transformations for Petri Net Technology.* PhD thesis, Technical University Berlin, 2003.

[18] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Proc. 19th International Conference on Application and Theory of Petri Nets*, LNCS 1420, pages 1–25. Springer, 1998.

[19] R. Valk. Concurrency in Communicating Object Petri Nets. In G. Agha, F. de Cindio, and G. Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, LNCS 2001, pages 164–195. Springer, 2001.

A Appendix

Specifications of (labeled) directed graphs, graph morphisms, and graph category in HasCasl:

spec LABELEDGRAPH =

then

var $N, E, L : Type$

type $LGraph\ N\ E\ L = \{(n, source, target, nlabel, elabel) :$
$Set\ N \times (E \rightarrow ?\ N) \times (E \rightarrow ?\ N) \times (N \rightarrow ?\ L) \times (E \rightarrow ?\ L) \bullet$
$dom\ source = dom\ target$
$\wedge\ (source :: dom\ source \longrightarrow n)$
$\wedge\ (target :: dom\ target \longrightarrow n)$
$\wedge\ cod\ source = dom\ nlabel$
$\wedge\ dom\ source = dom\ elabel\}$

ops $nodes : LGraph\ N\ E\ L \rightarrow Set\ N;$
$edges : LGraph\ N\ E\ L \rightarrow Set\ E;$
$sourceMap, targetMap : LGraph\ N\ E\ L \rightarrow (E \rightarrow ?\ N);$
$nlabelMap : LGraph\ N\ E\ L \rightarrow (N \rightarrow ?\ L);$
$elabelMap : LGraph\ N\ E\ L \rightarrow (E \rightarrow ?\ L)$

forall ...

spec LGRAPHHOMOMORPHISM = LABELEDGRAPH
then

var $N1, E1, L1, N2, E2, L2, N3, E3, L3 : Type$
type $Hom\ N1\ E1\ N2\ E2 = \{(g1, hn, he, g2) :$
$LGraph\ N1\ E1\ L1 \times (N1 \rightarrow ?\ N2) \times (E1 \rightarrow ?\ E2)$
$\times\ LGraph\ N2\ E2\ L2 \bullet$
$hn :: nodes\ g1 \longrightarrow nodes\ g2 \wedge he :: edges\ g1 \longrightarrow edges\ g2$
$\wedge\ \forall e : E1 \bullet e\ isIn\ edges\ g1 \Rightarrow$
$(hn(sourceMap\ g1\ e) = sourceMap\ g2(he\ e)$
$\wedge\ hn(targetMap\ g1\ e) = targetMap\ g2(he\ e)$
$\wedge\ elabel\ g2(he\ e) = elabel(g1\ e)$
$\wedge\ \forall n : N1 \bullet n\ isIn\ nodes\ g1 \Rightarrow$
$nlabel\ g2(hn\ n) = nlabel(g1n)\}$
type $LHomHom\ E\ N := LHom\ E\ N\ E\ N$
ops $dom : LHom\ N1\ E1\ N2\ E2 \rightarrow LGraph\ N1\ E1\ L1;$
$cod : LHom\ N1\ E1\ N2\ E2 \rightarrow LGraph\ N2\ E2\ L2;$
$nodeMap : LHom\ N1\ E1\ N2\ E2 \rightarrow (N1 \rightarrow ?\ N2);$
$edgeMap : LHom\ N1\ E1\ N2\ E2 \rightarrow (E1 \rightarrow ?\ E2);$
$id : LGraph\ N1\ E1\ L1 \rightarrow LHom\ N1\ E1\ N1\ E1;$
$\_o\_ : LHom\ N2\ E2\ N3\ E3 \times LHom\ N1\ E1\ N2\ E2$
$\rightarrow ?\ LHom\ N1\ E1\ N3\ E3$
pred $injective : LHom\ N1\ E1\ N2\ E2$
forall ...

spec LGRAPHCATEGORY = LGRAPHHOMOMORPHISM **and**

sorts N, E, L
then

types $G := LGraph\ N\ E\ L;$
$H := LHomHom\ N\ E$
$M = \{h : H \bullet injective\ h\}$

view CATEGORYOFLABELEDGRAPHS : HLRCATEGORY
to LGRAPHCATEGORY =
$Ob \mapsto G, Mor \mapsto H, \_o\_, dom, cod, id, M$